

Tecnológico de Costa Rica

High Performance Embedded Systems

Project 3: Optimizing Traditional and Deep Learning Algorithms for  
Autonomous Driving

Report

Professors

MSc. José Araya Martínez  
MSc. Sergio Arriola-Valverde

Students

Sergio Guillén Fernández  
Eliécer Mora Alaniz

Second Third of year  
2020

## SGBM Algorithm Profiling

Given that the result obtained when attempting to profile with valgrind the provided implementation of the SGBM algorithm is that it takes a lot of time just to start the the Graphical User Interface (GUI) in the provided implementation, it was required to port the implementation to a console-based application. After implementing this, it was possible to obtain good results in the profiling. Part of the most important results are presented in the Illustration 1.

Function	Location	Called	Self Cost: Ir	Incl. Cost: Ir
0x00000000000011c0	/lib/aa...	0	2	957 876 396
dl_start	/build/...	1	308	957 876 394
main	/build/...	222	13 427 553	957 876 393
dl_init	/build/...	1	22	954 182 010
0x000000000000381c	/home/...	1	17	951 163 919
(below main)	/build/...	1	31	951 163 902
_libc_csu_init	/home/...	1	62	951 163 794
main	/home/...	1	22	951 162 022
void std::cx11::basic_strin...	/home/...	1	33	951 003 159
StereoFGA::cost_computati...	/home/...	2459317124	682 794	552 466 656
StereoFGA::find_minLr(int*)	/home/...	2458528	410 574 176	410 574 176
StereoFGA::compute_hamm...	/home/...	1	11 950 774	325 760 182
StereoFGA::compute_hamm...	/home/...	692736	313 809 408	313 809 408
StereoFGA::cost_aggregatio...	/home/...	1	25 123 305	25 123 305
StereoFGA::compute_censu...	/home/...	2	20 028 688	20 028 688
find_min(int, int, int, int)	/home/...	2458528	17 209 696	17 209 696
StereoFGA::init_Lr(int*, int*)	/home/...	1	12 418 595	12 418 595
StereoFGA::calc_disp(int*, c...	/home/...	25872	7 043 504	7 407 171
dl_start	/build/...	1	512	3 694 076
dl_start	/build/...	1	24	3 693 564
dl_start*2	/build/...	12	306	3 693 540
dl_start*2	/build/...	1	6	3 693 234
dl_start_final	/build/...	1	48	3 693 228
dl_sysdep_start	/build/...	1	404	3 693 046
dl_sysdep_start*2	/build/...	3	26	3 692 542

Illustration 1: Results after profiling the SGBM algorithm.

Given this, it was considered that the three most computation-expensive functions are:

- cost\_computation.
- find\_minLr.
- compute\_hamming\_distance.

## Memory optimization

Once the algorithm was profiled, the next step was to optimize the amount of memory based on some assumptions in the input parameters.

Assuming that the input parameters are restricted to:

- BlockSize: 5 pixels
- P2 : 128
- NumDir: 4
- $C_{\max}(p, d)$ : Results from Equation 3

The following results were obtained:

For equation 2:

$$\begin{aligned}\text{Census Transform Bit depth} &= \text{Blocksize} * \text{Blocksize} - 1 \\ &= 5 * 5 - 1 \\ &= 24\end{aligned}$$

This allows to reduce the size of the array that holds the census transform from 64 bit elements to 32 bits elements. This is: ct1 and ct2 in the implementation. This implies changing the declaration and implementation of the functions:

- compute\_census\_transform
- compute\_hamming
- compute\_hamming\_distance

Also, it is possible to reduce the size of the m\_u8BlockSize\_half to one byte as the maximum value stored here fits in that size.

For equation 3:

$$\begin{aligned} C_{\text{m}\acute{\text{a}}\text{x}}(p, d) &= \text{Log}_2(\text{Census Transform Bit depth}) \\ &= \text{Log}_2(24) \\ &= 4.58 \end{aligned}$$

Given than this equation provides the result of the bit depth after computing the hamming distance. So, we are changing from a theoretical bit depth of 24 (32 in reality) to one of 4.58 bits (8 in the implementation). This allows to change the size of the elements the application reserves for the hamming distance computation such that they are modified from an integer size (32 bits) to an unsigned integer of 8 bits. Specifically, the type of m\_ActiveInitCost was changed from int to uint8\_t.

This implied a change in the declaration and implementation of the following functions:

- compute\_hamming
- compute\_hamming\_distance
- init\_Lr
- compute\_SGM

For equation 4:

$$\begin{aligned} L &\leq C_{\text{m}\acute{\text{a}}\text{x}}(p, d) + P2 \\ L &\leq 4.58 + 128 \\ L &\leq 132.58 \end{aligned}$$

As this result provides the higher value we'll get for the cost computation, we conclude the size of the reserved elements for the cost computation can be reduced from an integer to an integer of 8 bits. Concretely, the type of the pointer m\_ActiveLrCost is set to an int8\_t.

All the related function were updated to use this type.

For equation 6:

$$\begin{aligned} S &\leq \text{Num. Dir} * (C_{\text{m}\acute{\text{a}}\text{x}}(p, d) + P2) \\ S &\leq 4 * 132.58 \\ S &\leq 530.32 \end{aligned}$$

Finally, this compute value allows us to change the type used for the aggregate in the cost function such that it is changed from an integer to an unsigned integer of 16 bits.

## Multi-threading

The initial execution time for the `cost_computation` function is around 12.798 s, for the `find_minLri` it is around 242 us. This time is really small, however, as can be seen in Illustration 1, it is called a lot of times and that's why it was selected. Finally, the function `compute_hamming` takes around 2.386 s without any optimization.

The results obtained after attempting to optimize the mentioned functions in the profiling section of this report is summarized in the table 1.

Function	Initial execution time (s)	Optimized execution time (s)
<code>cost_computation</code>	12.798	3.33
<code>find_minLri</code>	242*E-6	242*E-6
<code>compute_hamming_distance</code>	2.386	2.378

Table 1. Summary of the initial and optimized execution times

As seen in the table, there are some functions that were not optimized. It was attempted to reduce the execution time, however, the obtained result is that the execution time was increasing as the threads augmented. In the case of the hamming distance, the result was similar but the execution time did not increase as much as with the `find_minLri` function.